Issue 34 | Jul 2011

blender art MAGAZINE

Blender learning made easy...

Rigs, Addons and more!

www.blenderart.org

Monstro -by Carlos Fernando

CONTENTS

EDITOR - Gaurav Nawani
MANAGER/EDITOR - Sandra Gilbert
WEBSITE - Nam Pham
DESIGN - Gaurav Nawani

PROOFERS
Brian C. Treacy
Bruce Westfall
Daniel Hand
Daniel Mate
Henriël Veldtmann
Joshua Leung
Joshua Scotton
Kevin Braun
Mark Warren
Noah Summers
Patrick ODonnell
Phillip
Ronan Posnic
Scott Hill
Wade Bick
Valérie Hambert

WRITERS
Oscar Baechler
Alexiss Dawn Memmott
Juan Pablo Bouza
Bart Crouch
Pratik Solanki
Pedro Bastos
Xenxo Alvarez
Veronica Orvalho
Osman Acasio
Chaiwit Jarunyakorn
Sandra Gilbert
Gaurav Nawani

COVER ART
Monstro -by Carlos Fernando

Upcomming Issue 35 - "Character Building"

- Possible Character article topics for you to submit:
- Making of/tutorials on modeling characters
- Articles about current projects that involve characters
- Topology tutorials
- Lo poly Characters
- Texturing/Mapping of characters

EDITORIAL

Sandra Gilbert

Manager/Editor

"When I am required to rig a character I run into problems one after another and eventually call it good long before it really is."

There are many things I love doing in Blender. Rigging is not one of them. Which of course means I have put as little effort into learning the finer points of rigging as humanly possible.

This attitude does cause more than a little frustration when I am required to rig a character. I run into problems one after another and eventually call it good long before it really is. So far this has only caused minor annoyance, because I generally stick to still images. And I have gotten really good at camouflaging nasty rigging errors. But this just will not work for any type of animation. Sigh, I guess it is time to bite the bullet and pay attention to all the wonderful tutorials and resources currently available for reluctant riggers like me.

Luckily there is an abundance of tutorials on rigging. These days you can find numerous tutorials that cover very simple rigs to very advanced and complicated rigs that almost animate themselves (Okay not really, but that would be nice).

And for those that want an even quicker option, there are a few add-ons to make short work of creating a usable rig.

We have gathered a wonderful collection of tutorials to help you learn the finer points of rigging, leaving you with no excuses to avoid it any longer.

So cozy on up and let's get started ●

Izzy Speaks: Turn Off the Sound

Izzy Speaks

Turn off the sound

Recently I read that one way to determine if your animation works is to turn off the sound. Can you still follow the story?

A few days later, with that tip floating around my subconscious, I hear my four year old laughing herself silly. She was watching Pink Panther. I remember watching Pink Panther when I was a kid and laughing myself silly just like she was doing. Each episode was filled with things going wrong for the inspector and the Pink Panther just seemed to always come out on top.

I sat down to watch with my four year old and realized that not only is Pink Panther still funny, it is an excellent tool / resource for studying animation techniques.

Remember that "turn off the sound tip"? Here is a perfect example of that tip in practice. There is almost no dialog and yet even a four year old can follow along with the story.

Lines of action are clear, the 10 principles of animation are used to great effect, silhouettes are boldly drawn and the characters have very expressive faces and body actions. This is classic animation at its best.

So google Pink Panther and sit on down for some hysterically funny studying.

Character Creation Vol.2 – Rigging

Lee Salvemini returns with Volume 2 of his Character Creation series. The second installment will take the modeled Ninja and make it animation ready. This is truly a complete reference for rigging that you can come back to again at any point during your time learning Blender.

How to use face tracking data in Blender

Sebastian König has posted a video tutorial on Vimeo showing how easy it is to use face tracking data from easycapstudio.com in Blender. In addition to the video, Sebastian has posted a link to get tracking data from easycapstudio.com as well as a link for the model he used in the tutorial.

cmiVFX Tutorial: Massive Mammoth Masterclass

Sebastian König's Mammoth gets rigged for animation at cmiVFX with the help of Nathan Vegdahl.

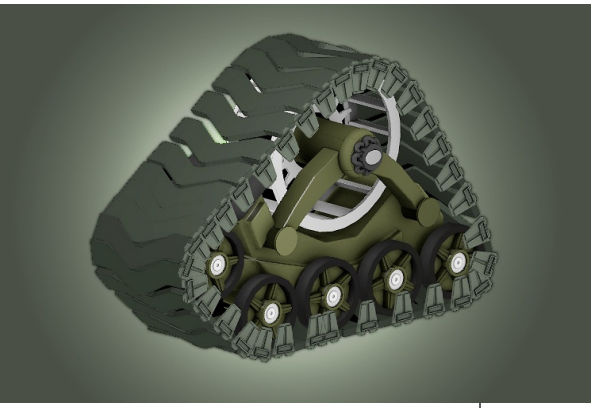
There is over 6 hours of content, divided into a two part masterclass covering the secrets of perfect weighting, efficient control rigs and elegant shape keys.

The first part will cover some essential rigging concepts, the setup of the deformation rig and creating an animator-friendly control rig with custom bone shapes.

The second part of this tutorial (originally part of the first video) covers weighting, corrective shape keys and non-corrective shape keys.

Blender Cookie has excellent rigging tutorials covering a variety of rig types, tips and usages. You can find the tutorials in the Animation section. Most are free, a few like the Baker series are citizen exclusives. These are a great investment in your Blender education

3D WORKSHOP : Tank Treads Rig for Game



by - **Chaiwit Jarunyakorn**
(Graphicianz)

Introduction

One of the classic problems in 3D is how to rig tank treads. It's sort of like a prime number problem in programming. It's easy to rig the tank tread in an animation with Blender, but it does not work in-game because the constraints in animation and games are different. I'm going to show you how to create a tank tread rigging system with a python script for the Blender Game Engine.

The solution is a little bit complex because some useful constraints do not work in-game. This means I have to create all the movement and rotation of each tread with the python script.

Main Idea

Create one object to be a path of movement (this should be a mesh, not a curve). The object must have many vertices because the movement of each tread part will read from the vertices of the path.

Read the coordinates of the vertices in the path object and store this value to a list variable.

Create many tread parts and make them run along a path. The distance of each tread part will be about 15-20 vertices of the path.

Make each tread part rotatable by finding the slope of the current vertex that the tread is on and the next vertex in the path object. Then convert the slope to a value in degrees and rotate each tread part with this value.

Let's start

Create an engine and wheel object to place inside the tank treads.

Figure 1 – The engine and wheel of the tank

Next, create an object to be the path by going to Add / Mesh / Circle. I set vertices to 500 and named this object "line". Then, modify it in edit mode for each tread part that will be running along the object.

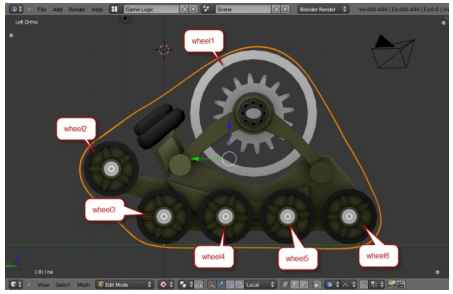


Figure 2 – The path that the tank tread will run along

Make each wheel the same object by using the join command and rename them to "wheel1" , "wheel2" , "wheel3" , ... (as shown in the image). Then, make "wheel1" rotatable by creating 3 states

State 1 : no rotation

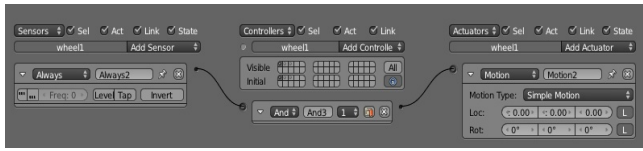


Figure 3 – Logic brick in state 1

State 2 : rotate along X-Axis for -1˚C (forward rotation)

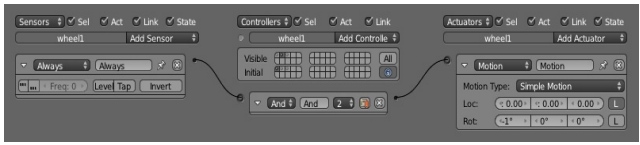


Figure 4 – Logic brick in state 2

3D WORKSHOP : Tank Treads Rig for Game

State 3 : rotate along X-Axis for 1˚ C
(backward rotation)

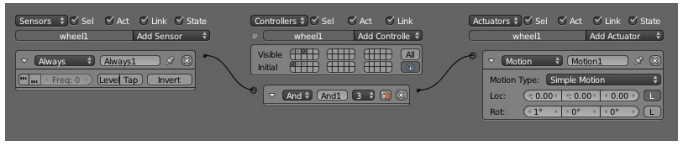


Figure 5 – Logic brick in state 3

Copy all of the logic bricks from “wheel1” to the others by selecting “wheel2” to “wheel6”, and then select “wheel1” last. Go to Object / Game / Copy Logic Bricks. So, if I want these wheels to stop, rotate forward or rotate backward, I just change their state to 1, 2, 3 by python script or doing it directly.

One characteristic of tank treads is that they are like a small piece of chain that connects together. For the example piece, the top is on the Z axis and the side is on the Y axis. You should place a small piece on the same axis that I have. Otherwise, you'll have a problem during rotation because I'll refer to these axes when I rotate each piece in the python script. Rename the tread to “tread_part.000”.

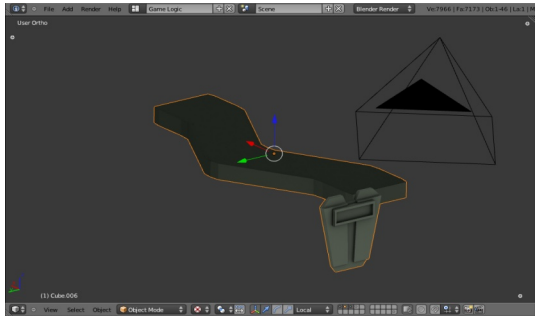


Figure 6 – The tread part

Now create a script named "readVerts.py". I use this script to read all the vertices from the "line" object. The vertices have a vector data type that stores x,y and z coordinates. Many vertices are stored as a "list" variable, but each coordinate is a local coordinate of the "line" object. In that case, if I move a path to anywhere in the scene, the coordinate that I read is still the same, but I need something different. What I need is the coordinate data of each vertex in the path object that references from global (world). Therefore, each vertex that is read from the path object (local coordinate) just needs to be summed to the path position to give the global coordinate. By the way, all coordinates will be stored as “list” variables named “verts”.

```
##readVerts.py
from bge import logic
import bpy

cont = logic.getCurrentController()
obj = logic.getSceneList()[0].objects
owner = obj['line']
logic.verts = []
curPos = owner.position
logic.linePos = curPos
for item in bpy.data.objects :
    #print(item.name)
    if item.type=='MESH' and item.name=='line':
        for vertex in item.data.vertices:
            logic.verts.append(vertex.co+curPos)
```

Make the “line” object call the “readVerts.py” script (Don’t forget to make this sensor call this script repeatedly by pressing the “...” button in front of the sensor, otherwise the coordinate data will not update if you call the script just once when the path moves).

Create a script called “control.py” which has these 3 functions:

* initMove() function : used to initialize movement state.

* key() function : used to receive up arrow and down arrow key from user to update movement state.

* addPart2Scene() function : use to read each tank tread from layer 2 into layer 1

```
#control.py
from bge import logic

def initMove():
    logic.move = "stop"
    #print("init move")

def key():
    cont = logic.getCurrentController()
    uparrow = cont.sensors['uparrow']
    downarrow = cont.sensors['downarrow']

    if logic.setuppart != "NOT_FINISH":
        if uparrow.positive :
            logic.move = "forward"
            for obj in logic.wheel:
                obj.state = 2
```

3D WORKSHOP : Tank Treads Rig for Game

```
        elif downarrow.positive:
            logic.move = "backward"
            for obj in logic.wheel:
                obj.state = 4
        else :
            logic.move = "stop"
            for obj in logic.wheel:
                obj.state = 1


def addPart2Scene():
    cont = logic.getCurrentController()
    owner = cont.owner
    i = owner['i']
    if i<33:
        addobj = cont.actuators['addobj']
        addobj.object = "tread_part."+str(i).zfill(3)
        cont.activate(addobj)
        owner['i'] = i+1
    else :
        logic.setuppart = "FINISH"
```

Create a script called “initwheel.py”. I use this script to read all the wheel objects a into list variable called “wheel” (line 6 from the python code, I use a short term assignment with list variables by using a “for loop” to read all objects whose names start with “wheel”).

```
#initwheel.py
from bge import logic

#init wheel
obj = logic.getSceneList()[0].objects
logic.wheel = [part for part in obj if part.name.startswith('wheel')]
logic.setuppart = "NOT_FINISH"
```

Next, connect a logic brick into the “line” object as shown in Figure 7

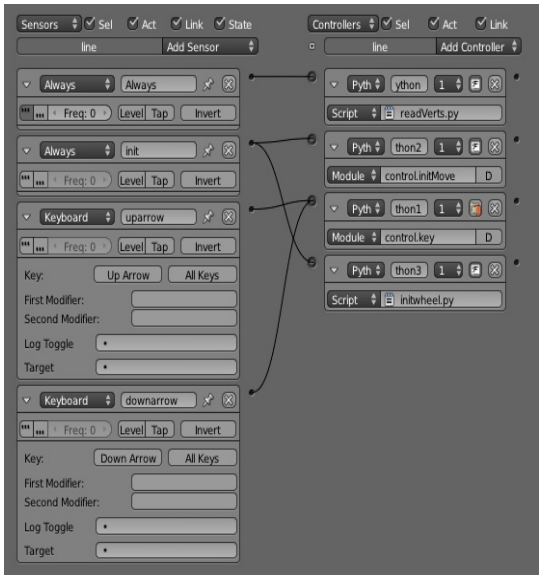


Figure 7 – “Line” object’s logic brick

Create a script called “initpart.py”. I use this script to initialize the variable “i” by reading the last 2 positions of an object by using its name which is used to assign the position of each tread part (this is important as I’ll eventually create 32 duplicates of the “tread_part.000” whose names will be “tread_part.000” through “tread_part.032”). After this, assume i=0 to make the tutorial easier to understand.

First, look at lines 17-20. This is where I initialize the tread part position by reading verts[0] and assigning i1=i , i2=i+1. In lines 22-31, I make each tread part rotatable when running along the curve of the path. The problem is “How many degrees does that tread part have to rotate in each position of the curve?” The solution is to find the slope of the line between the current vertex and the next one, convert the slope into radians (m = arctanθ), and finally convert the radians to degrees.

Now let’s examine lines 34-40. The tread part had the wrong rotation at some point in the path because it can only rotate in quatrain 1 and 2. The solution is to add or subtract 180 at the point that had the wrong rotation by checking the localY and localZ variable.

Finally, line 42. Rotate the tread part by an amount of “degrees”. So each tread part calls this script only once when starting up.

```
#initpart.py
from bge import logic
from math import radians,atan,degrees
import mathutils

cont = logic.getCurrentController()
owner = cont.owner
initpart = owner['initpart']

#init i from object's name
n = int(owner.name[-2:])
owner['i'] = n*15

if hasattr(logic, 'verts') and initpart==0:
```

3D WORKSHOP : Tank Treads Rig for Game

```
try:
    #init position and rotation of tread part
    owner.position = logic.verts[owner['i']]
    i1 = owner['i']
    i2 = owner['i']+1

    z1 = logic.verts[i1][2]
    z2 = logic.verts[i2][2]
    y1 = logic.verts[i1][1]
    y2 = logic.verts[i2][1]
    m = (z1-z2)/(y1-y2)

    deg = degrees(atan(m))

    localZ = z1-logic.linePos[2]
    localY = y1-logic.linePos[1]

    #head part
    if localZ<owner['localZ_head'] and localY<2.1849:
        deg = deg+180
        #print("add")
    #foot part
    if (localY>-5.64 and localY<=-5.14) and
localZ>owner['localZ_foot']:
        deg = deg-180
        #print("minus")

    owner.localOrientation =
mathutils.Matrix.Rotation(radians(deg), 3, 'X')
  except :
    pass
```

Create a script called “followline.py”. This script has functions like the“initpart.py” script but the difference is that the tread part will call this script repeatedly and check whether to run forward or run backward.

Make the “tread_part.000” object call the “followline.py” and “initpart.py” scripts by creating a logicbrick and adding the properties shown in Figure 8

```
#followline.py
from bge import logic
from math import radians,atan,degrees
import mathutils


cont = logic.getCurrentController()
owner = cont.owner
i = owner['i']
m = 0

if logic.move == "forward" or logic.move ==
"backward":
  if hasattr(logic, 'verts'):
    owner.position = logic.verts[i]

    if logic.move == "forward":
      #forward moving
      if i>(-len(logic.verts)+1):
        owner['i'] = i-1
      else :
        owner['i'] = 0
      owner['localZ_head']=1.99
      owner['localZ_foot']=0.64
      #print("move = ",logic.move)
    elif logic.move == "backward":
      #backward moving
      if i<len(logic.verts)-1:
        owner['i'] = i+1
      else :
        owner['i'] = 0
      owner['localZ_head']=1.96
      owner['localZ_foot']=0.644
      #print("move = ",logic.move)

  #rotate
  #i is old i
  #owner['i'] is new i
  #find m (slope)
  try:
    z1 = logic.verts[i][2]
    z2 = logic.verts[owner['i']][2]
    y1 = logic.verts[i][1]
    y2 = logic.verts[owner['i']][1]
    m = (z1-z2)/(y1-y2)

    deg = degrees(atan(m))

    localZ = z1-logic.linePos[2]
    localY = y1-logic.linePos[1]

    #condition for both
    if localZ<owner['localZ_head'] and
localY<2.1849:
      deg = deg+180
      #print("add")
    #foot part
    if (localY>-5.64 and localY<=-5.14)
and localZ>owner['localZ_foot']:
      deg = deg-180
      #print("minus")

    owner.localOrientation =
mathutils.Matrix.Rotation(radians(deg), 3,
'X')
  except:
    print("follow line")
```

3D WORKSHOP : Tank Treads Rig for Game

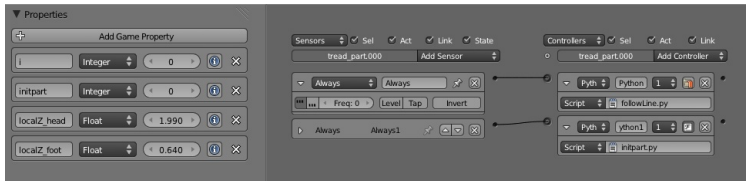


Figure 8 – "Tread_part.000" object's logic brick

Next, I've duplicated the "tread_part.000" object until I had 33 objects. You can either press duplicate (shift+d) 32 times or you can do this via a python script.

Create a script called "inittread.py". When I press spacebar, this script will duplicate the tread part 32 times. When released, the script will delete the logic brick set that calls this script (because I use this script only once). Otherwise I just have to delete the logic bricks that call this script manually for 33 objects.

```
#inittread.py
import bpy
from bge import logic

cont = logic.getCurrentController()
space = cont.sensors['space']

def copyTread():
    if space.positive:
        #copy 33 objects
        for i in range(1,33):
            bpy.ops.object.duplicate()
        print("create 33 objs")
    else :
        #logicbrick remove
        for i in range(0,33):
            bpy.ops.logic.sensor_remove
(sensor="space",object="tread_part."+str(i).zfill(3))
            bpy.ops.logic.controller_remove
(controller="script",object="tread_part."+str(i).zfill(3))
        print("remove logic brick")
```

After running the script "inittread.py", I'll get 32 duplicates of the "tread_part.000" (giving me a total of 33 objects) named "tread_part.000" through to "tread_part.032". Next, move all the tread sets to layer 2.

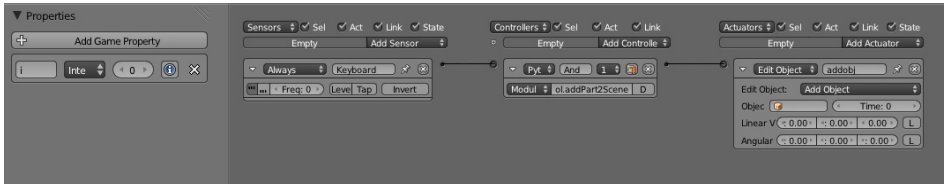


Figure 9 – The "Empty" object's logic brick

Go to layer 1, create an Empty object and make the logic brick call the module "addPart2Scene" in the "control.py" script. This will read each tread part in layer 2 and assign it to an "Edit Object" actuator in "Add Object" mode for 33 objects.

Finally, make the logic brick connection as shown in Figure 9 ●

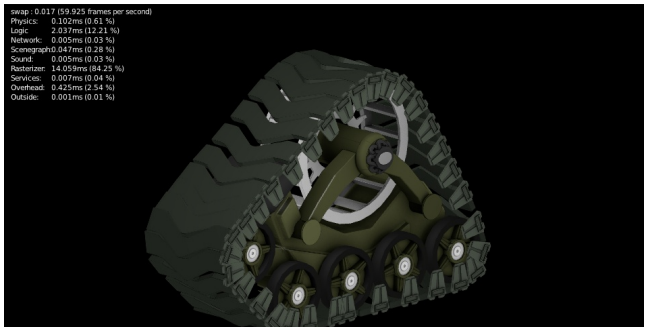


Figure 10 – Showing the final result

3D WORKSHOP : Basic Rigging of a Fish

by - **Osman Acasio**

Introduction

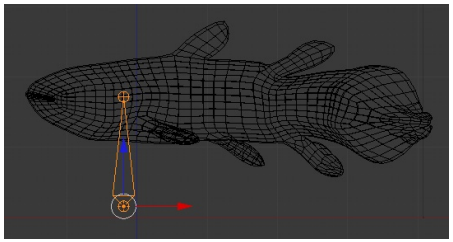
The most practical way to learn how to rig 3D models in Blender is to start with a basic and simple exercise. The following is a tutorial written for novices that want to learn this beautiful art. Rigging a fish isn't that difficult and this will give us the basic knowledge to utilize the technique.

We begin by loading our scene with a fish model created previously for this tutorial. Do this by going to File>>Append (Shift F1) and look for the file: "Fish.blend" where we can find the 3D creation. To make it more interesting, I opted to model a Celacanto, a true living fossil. Once loaded, we save the scene.

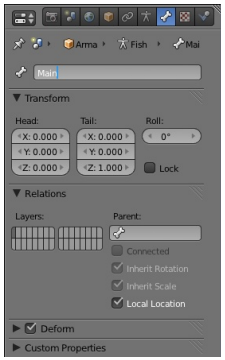
It is now time to add a Bone for our fish which we do by going Add>> Armature>> Single Bone. The Bone has a body in the form of an Octahedron, a root (represented by a ball at the wider end of the body), and the tip (represented by a ball at the narrow end of the body). Both the tip and root can be selected separately. The whole bone can be selected by clicking in the centre of the Bone.

Move the bone in such a way that the tip is amid the head of the fish, behind its eyes as shown in the picture.

Give a name to the Bone so you can

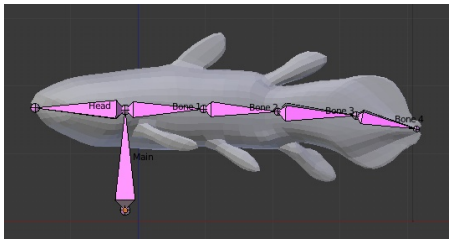


identify it from the rest of your Bones. "Main" would be an appropriate name. In the panel on your right, look for the Bone panel represented by the small bone. There will be a field there to allow to you change the name of this bone from "Bone" to "Main."

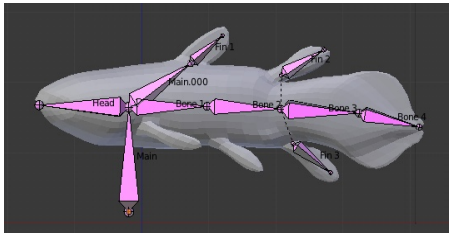


Let us continue adding bones to our scene. This time we will make them by means of extrusion. Select the tip of the main Bone and with the E key, extrude the Bone that will move the head. Change the name of this Bone to "Head".

We select the tip of the "Main" Bone again and extrude another bone away from the "Head" Bone with the E key. We repeat the process until we have given the body of the fish several Bones that end at the tail (four in total). We assign the names: "Bone 1" through to "Bone 4" for these Bones.



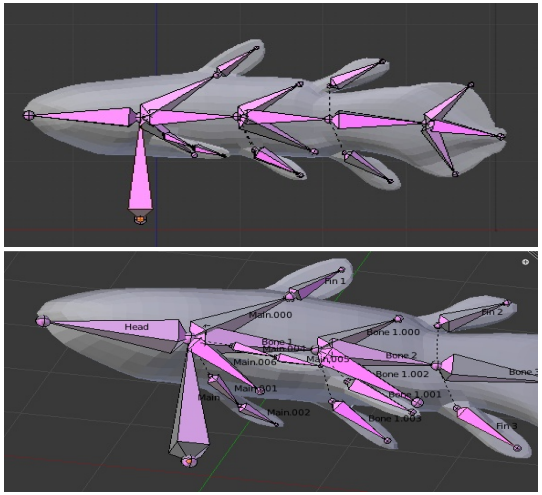
From the Main Bone extrude one Bone, then another to give us the Bone that will move the front upper fin. We will call this second Bone "Fin 1". From Bone 2 extrude two sets of Bones that will connect with the Bones that will work the rear upper and rear lower fins. Name these Bones "Fin 2" and "Fin 3" respectively. Notice that we eliminate the Bones that connect Bone 2 with the Fin 2 and 3 (see figure).



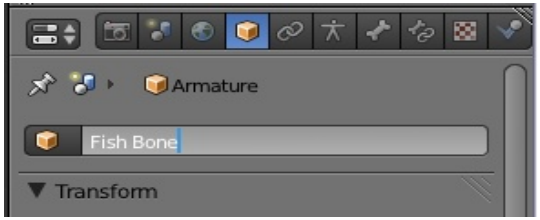
We assign the other bones to the remaining fins in a similar way and

3D WORKSHOP : Basic Rigging of a Fish

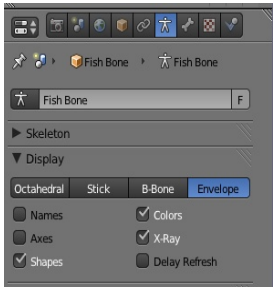
place them in the appropriate locations. Make sure you select the intermediate Bones between the fins and the Main Bone and eliminate them one by one. From the tip of Bone 1, extrude two more Bones for the body. From the tip of Bone 3, extrude two more Bones for the fins of the tail. The result should look similar to the following image.



It is now time to give a name to our skeleton. We go to the Object panel represented by an orange cube. In the field that shows "Armature" we substitute it for "Fish Bone." A quite appropriate name.

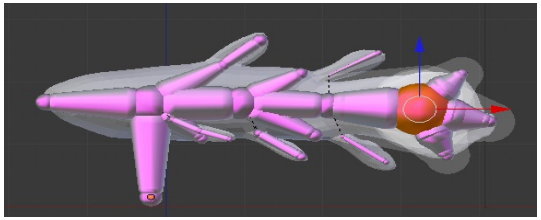


With all the bones in their place, we now have to prepare the mesh of the fish to unite it to the skeleton. It is necessary to abandon the Edit Mode through [TAB]. Select the mesh of the Celacanto, and look for the "Modifiers" panel (represented by a wrench). In "Add Modifiers" we assign the option Armature and in the field object we select "Fish Bone".



After this procedure, when you select the skeleton while in Pose mode, moving a bone will move the skin of the fish. Some imperfections can be corrected easily by toggling the "Envelope" option in the Armature panel.

We adjust the envelope in such a way that it covers the whole geometry of the fish by scaling the root of the bones with the S key in Edit Mode.



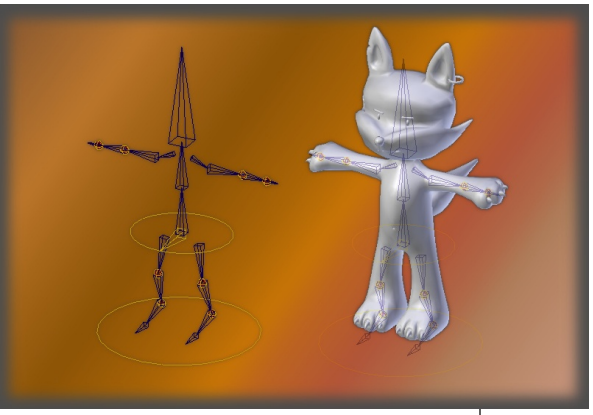
If done correctly, the envelopes will look like tin the figure.

Our clever fish can now recover life through the technique of animation and swim and travel in underwater environments generated by computer ●

Osman Acasio. Lives in Venezuela. Is an Industrial Engineer specializing in 3d projects. He has modelled industrial creations for diverse industries and produced animation in Acasio Infográfica. Uses Blender alongside 2D and 3D CAD packages to both quickly prototype new products and assemblies. Found Blender an extremely productive tool to be accepted as an alternative engineering design tool for model engineers.

www.proyectoacasio.blogspot.com

3D WORKSHOP : Building a Switchless and Intuitive Body Rig



by - **Pedro Bastos , Xenxo Alvarez & Veronica Orvalho**

Introduction

In this article we describe the implementation process in Blender of a new intuitive body rig we developed. Our goal is to provide an alternative to the common character animation techniques. For instance, our rig is very animator-friendly because it mimics forward and inverse kinematics with no need for FK/IK control switches or FK/IK matching. It also allows character posing as if doing drag and drop, causing an augmented feeling of control in animation.

Method

The rig works by freely selecting, dragging and positioning the controls of the hip, waist, hands, elbows, feet and knees. Rotation is only used in the joints of the hip, waist, hands and feet to achieve specific poses. Figure 1 shows the 3D model of a character (1) previously developed in our research group (the Porto Interactive Center) which is now used for testing the body rig (2) we developed for testing purposes.

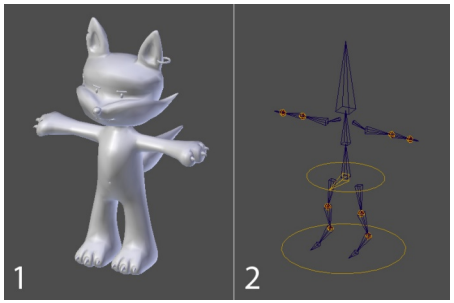


Figure 1: The 3D model of a character (1) and the intuitive body rig controlling it (2)

The hip, waist, spine and head use a single-bone based hierarchy. The arms and legs use two overlapping bone based hierarchies, one is the control bones and the other is the guide bones. The purpose of the first is to control the mesh of the character through skinning with squash and stretch abilities. The purpose of the second is to mimic the behavior of the first but not using any squash and stretch. The guide bones always maintain their initial proportion, which is anatomically correct. The goal is for the user to operate on the control bones (colored in yellow) using the guide bones (the joints colored in red) as a visual aid for anatomical character posing. The user can also do arm and forearm hinge rotation by operating on the purple bones.

Implementation

The control bones hierarchy is the one the user will be manipulating in order to animate the character. The 'elbow.L' bone allows positioning of the left elbow and 'wrist.L' allows positioning and rotating of the left hand. These will be used more frequently although the user can also rotate bones 'arm.L' and 'forearm.L' for a more accurate hinge. Table 1 shows the parent-child relationships for the control bones hierarchy seen in Figure 2.

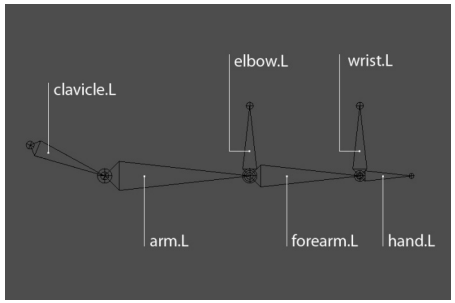


Figure 2: The left arm control bones hierarchy seen from the front view

Bone	Parent	Connected
clavicle.L	the last bone in the spine chain	No
arm.L	clavicle.L	Yes
forearm.L	elbow.L	No
hand.L	wrist.L	
elbow.L	the master bone of the character	
wrist.L		

Table 1: Relationships in the left arm control bones hierarchy

Following this, we assign the Stretch To constraint type to the 'arm.L' and

'forearm.L' bones making sure their targets are the 'elbow.L' and 'wrist.L' bones respectively. This allows the control bones to have squash and stretch abilities. Be sure to define the armature object in the constraint parameters in order to have access to the target bones, as seen in Figure 3.

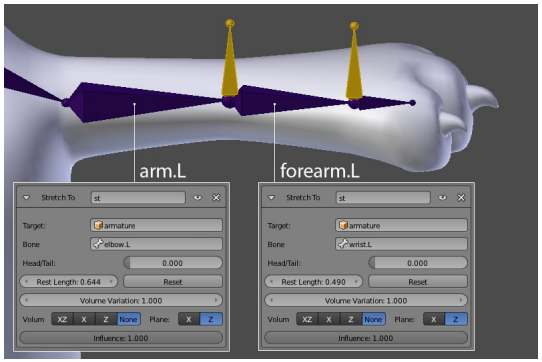


Figure 3: Constraints assigned to the left arm control bones hierarchy

Now we can move on to the creation and configuration of the guide bones in the rig. Figure 4 shows the left arm guide bones hierarchy seen from the front view. This hierarchy must match the control bones hierarchy (overlapping it). Notice that 'clavicle.L' bone is not to be duplicated and is shown in Figures 2 and 4 only as a visual reference for better visualizing the entire arm.

For naming the guide bones hierarchy we chose to use the prefixes 'Dist' and 'DistTip' because the purpose of this bone hierarchy is to let the user know the anatomical distance that should not be exceeded when manipulating the control bones. Table 2 shows the parent-child relationships for the guide bones hierarchy seen in Figure 4.

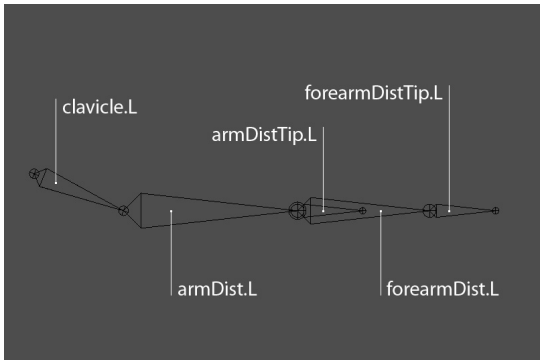


Figure 4: The left arm guide bones hierarchy seen from the front view

Bone	Parent	Connected
clavicle.L	the last bone in the spine chain	No
armDist.L	clavicle.L	Yes
armDistTip.L	armDist.L	
forearmDist.L	elbow.L	No
forearmDistTip.L	forearmDist.L	Yes

Table 2: Relationships in the left arm guide bones hierarchy

Following this, we assign the Copy Rotation constraint type to the 'armDist.L' and 'forearmDist.L' bones making sure their targets are the 'arm.L' and 'forearm.L' bones respectively as shown in Figure 5.

The setup in Figure 5 allows the guide bones hierarchy to mimic the rotation angles of the arm and forearm control bones hierarchy. But how will the user have a clear notion of the correct anatomical position of the elbow and wrist? This is due to the positions of the 'armDistTip.L' and 'forearmDistTip.L' bones in the guide bones hierarchy. These two bones are connected children of the 'armDist.L' and 'forearmDist.L' bones.

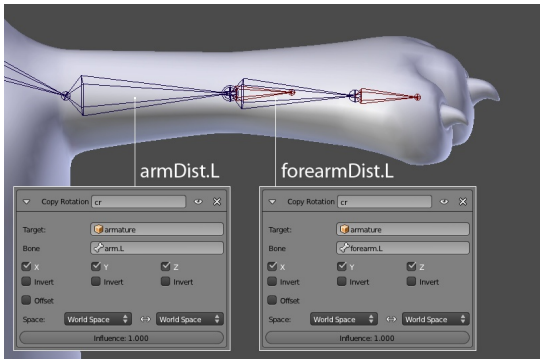


Figure 5: Constraints assigned to the left arm guide bones hierarchy

To keep the bones visually appealing for the animator we use custom shapes and three bone colors: yellow, purple and red. Figure 6 illustrates the end result for the left arm rig.

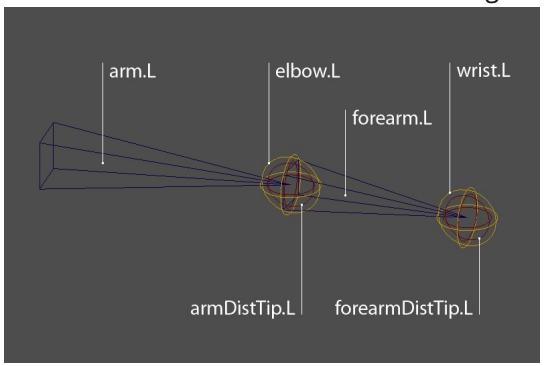


Figure 6: Custom shapes and bone colors used in the left arm rig

The skinning of the rig of the arm with the mesh of the character is done for only the following control bones in the rig: 'clavicle.L', 'arm.L', 'forearm.L' and 'hand.L'.

Conclusions

The body rig method described in this article is straightforward and flexible, causing a feeling of drag and drop operability. The rig is pleasant for the user because it is accessible and flexible, resembling the handling of a puppet. It requires the user to get adapted to it because it generates animation curves that the experienced animator may find different for tweaking. Nonetheless, we believe our control structure enriches character animation because it produces fast character animation results with augmented control. The rig is adaptable to different body morphologies with only minor adjustments. And the rig supports squash and stretch which is ideal for cartoon characters. Furthermore, the rig can easily be extended to the entire body, namely to fingers and toes.

We anticipate improvements such as constraining the control bones to the guide bones when the user may require enhanced joint positioning. We didn't worry too much about the custom shapes; X-Ray could be prevented to make it easier to understand the visual controls.

We provide a video showing animation tests of the rig and a walk cycle for the character that we were able to build in less time and with less effort than using common methods.

Acknowledgments

In parallel to this article we submitted a theoretical approach of using our body rig for motion capture to the VERE PhD Symposium, online at http://www.vereproject.eu/.

For this issue of the BlenderArt magazine we chose to have a highly technical and less theoretical approach; more specific for character animation. We thank Teresa Vieira for the character design concept. If you have any questions e-mail us at ptbbastos@gmail.com.

3D WORKSHOP : Timing and spacing

by - **Pratik Solanki**

Introduction

I found Blender 3 years ago and liked it very much. I found that Blender, as open source software has its own potential which other proprietary software doesn't have.

But for animation, I must say it doesn't matter which software you use until you know what to achieve and how to actually do it. To learn animation with Blender is damn easy, trust me..... you just need to use proper techniques.

In this very first article we are not going to talk about skilled character animation, we are going to talk about some very basic and beginner things. Because if you don't have your basics clear, you wont be able to do good quality animations.

So I have heard this many times:

* to do better animation you need good sketching skills
* to make it even better, you need good acting skills
* and to take it to another level you also need some good 2D animation skills

Aaahh! I don't even have one of these .... what are you talking about man?

If you do know these skills then it would be an advantage for you, but if you don't then there's no need to worry .... You can still animate like the experts ... the learning method will just be different ... that's it.

Before we start the animation, it's important to think how you will set up your mind and what you will need. Always think positive while working and get references and inspiration from some good movies and cartoons.

Back to the topic:

So basically what you will need to learn is:

1. timing and spacing
2. stretch and squash
3. arcs
4. follow through
5. moving hold
6. recoil
7. anticipation
8. overlap
9. line of action
10. breaking joints
11. exaggeration
12. expression

These are the main things, but don't be afraid. They all are related to each other so there will come a time where you never have to be reminded of all of these principles. You'll just do it with your own creativity and knowledge.

Let's start with timing and spacing as it's one of the most important principles. You can only go further after you have learned it perfectly, otherwise you'll mess up on the other principles too. Many people get confused by timing and spacing as they are different things. It's better to try some examples than just trying to understand it theoretically and getting confused.

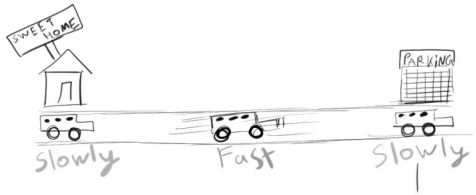
In every book and tut ... I mostly see this coin example:

It's very good but I honestly don't like that one, so here we will see the same technique but with a different

3D WORKSHOP : Timing and spacing

example.. hope this helps more.

In the diagram we have our car going from home to a parking lot.



This a very simple example, easy to understand and quite fun to learn. Imagine that you have your car at home and you want to drive it to a parking lot. What you would do is start your car, drive slowly out of the garage and then race your car on the open road like no one can stop you. Finally you brake to park your car at the parking lot and then slowly park in a parking space.

So that's how this thing happens. You now have one thing clear in your mind; how the car will move from one point (your home) to another (parking lot). Now let's decide timing.

Remember!

if the action is faster then your timing is short

if the action is slower then your timing is longer

Timing is something which you never understand from examples, so you have to try it based on your own sense of timing which is not that hard. To do that you can take a toy car or a ball and pass it from one point to another and note down that timing with a stopwatch. In this way you get your timing. See? It's easy and the next part is way more exciting than this one.

Spacing

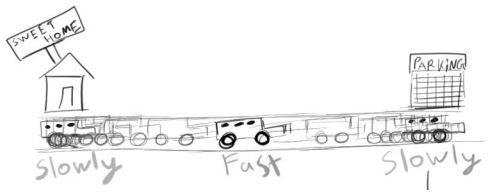
Now we will decide spacing. Spacing is one of the most important things in animation, but before we jump right in, keep in mind that we are not going to open Blender just yet. We will move to Blender after spacing has been decided.

How to decide spacing

Proper spacing gives your animation very good ease-in and ease-out. Sometimes it's very helpful to register poses and give weight to your animation.

Anyway, let's have a look at our car example. We have our very own car going from home to a parking lot. While working on your spacing, remember that whenever an object moves from one point to another point, it always starts with a slow speed then attains maximum speed at the middle and then stops with slow speed at the end.

In our car example it happens something like this:



The car starts slowly, gets up to speed and then eases in slowly at the end. But in this picture you can see lots of pictures when the car starts and stops and very few in the middle when the car has some speed.

From this we come to know that faster objects need very few keys and slower objects need lots of keys. That's how we did this in 2D.

But don't forget we have the power of 3D and we have a very powerful thing called Blender.

Let's have a look at the same example with a different perspective.

So now we use Blender and it means we don't need to setup this many keys, because all 3D software sets the in-between keys very well.

But we have all heard lots of times people saying that, "Hey don't ever trust the PC, it sets all the keys wrong. huh .. I hate that idiot box."

Look its not true everyone. If you make your PC do the wrong thing then it will, but if you put some effort in and proper keys on proper timing then you will surely be surprised by the end result.

Ok, now I'm using Blender but how will I set up my keys like that?

3D WORKSHOP : Timing and spacing

Well, you need some basic 2D knowledge to set your keys like that right?

Naaaaaaahh I don't think so, don't ever think like that. In 2D animation it's much harder than it seems. It will take lots of time to achieve a sense of spacing. But if you wanna make it in 3D we have some cool tricks to make it easier.

Let's assume we have a car model in Blender.

Just have a look at the spacing and forget timing for awhile. Set your 1st key on your 1st frame then move the car bit further and place another key on the 20th frame.

Now the car is traveling from one place to another place in 20 frames. Our 1st key and our 20th key are our contact poses.

A contact pose is how you learn what's going on in the story. It's a simple way of showing our story in pictures.

Ok, now it's time to add a breakdown key. A breakdown key is a key that gives your animation a flow. A breakdown key describes how to reach from one contact pose to another with a perfect arc and it lets your audience understand the movement better. Without it our animation looks flat and linear which we don't want.

So let's add a breakdown key.

Keep in mind that breakdown keys are always in the middle of two contact poses. So we have our breakdown key on the 10th frame.

After adding a breakdown key, let's add "in-between" keys that we will call IB.

IB keys gives your animation smoothness. It's very important for ease-in and ease-out and many other principles too.

Have a look at this image.

We have added IB keys on the 5th and 15th frame. IB keys are always in the middle of breakdown and contact keys. So now you have only five keys. See how easy it was.

I know, I know, you must say something like: "Hey man, it's not working, we are not getting that motion you described in the spacing section".

Ok so here is the trick.

Trick: *Think like you have magnet in every contact pose. This means your magnet affects your next pose. Remember I'm using the word "pose', the magnet affects the pose not the keyframe. This means we have a magnet on our 1st pose so it will affect the next one.*

So you started with something like this.

And now the magnet is attracting your next pose so it would be something like this.

So when we combine our frames with this image we will get something like this.

You can see that the pose on the 5th frame is attracted to the 1st pose and the same is happening with the next IB key (#15). It's attracted to the next contact (#20).

So in a simple way. On your 5th frame, set your pose in middle of the 1st and 5th frames distance and do the same thing with 15th and

3D WORKSHOP : Timing and spacing

20th. From this you will get surely fluid action in only 5 keys.

I hope this helps, remember practice this principle daily. Take 3-5 contact poses and think like they are magnets attracting all the IBs. You will surely learn a lot.

You might find this tutorial different from others, maybe because I'm self taught, but don't worry I'm learning it in my way and it always works for me and I hope it will work for you too.

In the next tutorial I will show you how to do a bouncing ball, because many people are always falling down at the bouncing ball. Just change your learning method and learn it in your way. You will definitely get how to do that. Or else I'll show you hehe. It's a very important thing for character animation ●

Hope you like this article.

cya

tc

Pratik Solanki

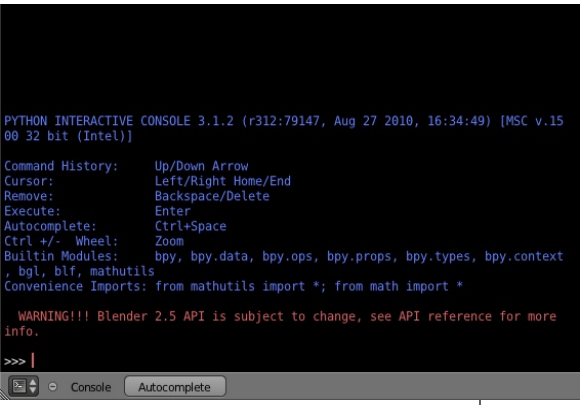
A little about myself. I am a very passionate CG artist seeking my path to animated success.

I have had some interest in this field from a very young age. I have been working on animation for 6-7 years and there's still a lot to learn and achieve for a more crazy animated future.

I am currently working as an animation director at Nifty Creation on an upcoming title.

www.dragoneex.com

3D WORKSHOP : Python scripting for artists



by - **Bart Crouch**

Introduction

As an artist you might think that python scripting is a complicated and boring task, but this is far from the truth. If you know a few basic principles it isn't very hard to do and it can actually save you time on doing boring tasks. Moreover, it even enables you to achieve results that are near-impossible without scripting.

This tutorial gives you a first introduction to python scripting in Blender and will enable you to start writing your own scripts. Therefore it will mainly focus on general methods of writing a script, which you can use in all kind of situations.

Note: if you wish to skip the first technical introduction you can start reading at Phase 2 and refer back to Phase 1 if there is something you don't understand. However if you have no experience with Python I suggest you start at Phase 1.

Phase 1 – first look at a script

The script we're going to write is a utility script for modellers. When modelling from a reference, it's useful to have a background image. Unfortunately Blender doesn't allow you to easily toggle between two or more opacity values for these images. So let's add this functionality ourselves.

> **Box1.** In Python the level of indentation is important. Changing the level of indentation will change, or even break, the script.
>
> An easy way to remember is that whenever you use a colon (:) the next line has to have an extra level of indentation. By convention we use 4 whitespaces as one level of indentation in Blender.

To start things, we have to set up a Blender scene that has some background figures. You can use [file1.blend] as a base to follow along with this tutorial. This file has two background figures, one for the top view and one for the front view.

We could start a new script from scratch, but let's make things easier for ourselves and use one of the templates that Blender has: **Text >> Script Templates >> Operator Simple**

Remove the last 2 lines, at which a test call is defined.

Let's take a moment to see how the script is structured and what it

```
1  import bpy                                                    1
2
3
4  def main(context):
5      for ob in context.scene.objects:                          2
6          print(ob)
7
8
9  class SimpleOperator(bpy.types.Operator):
10     '''Tooltip'''
11     bl_idname = "object.simple_operator"
12     bl_label = "Simple Object Operator"
13
14     @classmethod                                              3
15     def poll(cls, context):
16         return context.active_object != None
17
18     def execute(self, context):
19         main(context)
20         return {'FINISHED'}
21
22
23 def register():
24     bpy.utils.register_class(SimpleOperator)
25
26                                                               4
27 def unregister():
28     bpy.utils.unregister_class(SimpleOperator)
29
30
31 if __name__ == "__main__":                                    5
32     register()
```

Fig1 A python script

currently does. As shown in figure 1, the script can be divided into 5 parts.

Part 1 is needed to get access to Blender's functionality in Python. Below part 1 we get several code blocks that start with either class or def, respectively defining new classes and new functions. To understand the script, we can skip these blocks for the moment as they aren't called yet, they're just being defined.

This brings us to the final part (part 5), where you can see that if __name__ is equal to "__main__", the register() function is called. The important thing to remember is that in Blender this condition is only met when the script is run from the text-editor.

You can run the script by pressing the **Run Script** button in the header, or by pressing **[Alt + P]**.

When the condition in part 5 is met, it calls the register function, defined in part 4. This consists of the line:

```
bpy.utilis.register_class(SimpleOperator)
```

This calls the register_class function which is part of the utils module, which in turn is part of the bpy module (which we imported in part 1). This function registers our own operator, or panel or menu, with Blender. Enabling us to use our custom code inside Blender. The operator we make it register is defined in part 3.

Part 3 is where things get interesting. So far we've only seen code that is basically the same for all functionality you want to add to Blender, but what we write in our custom operator defines the new functionality. What follows is some explanation about the required parts for defining an operator:

The bpy.types.Operator argument (line 9) tells Python that we are subclassing from Blender's default operator struct. For instance, If you wanted to create a panel instead of an operator, you'd have to subclass from bpy.types.Panel.

bl_idname and bl_label are respectively the official name Blender uses inside its RNA structure and the name displayed to users.

The poll() definition is optional and defines if the operator can be used or not. If it returns a value of True, the operator can used. If it returns False, the operator can't be used. In this case it checks if there is an active object. For more information on Python syntax, like the comparison sign != , read box 3.

Finally the execute() function is executed when you call the operator. The final line (line 20) is obligatory, but before that you can do anything you want. In this case we call the main() function.

Box2: For easier code writing, enable the display of line numbers and syntax highlighting. This can be done in the header, see the image below.

To get feedback on errors it's useful to also enable the console. This can be done in the info header: Help >> Toggle System Console.

The current main() function in part 2 prints all objects in the scene to the console. This isn't what we want, but we'll change that in phase 2.

Let's first have a look at how we can test our code. The first critical step is to run the code **[Alt + P]**, if you haven't done so yet.

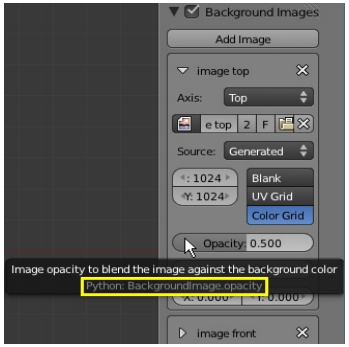
To run the operator, move your mouse to the 3d-view and open the search menu **[spacebar]**. Search for Simple Object Operator, the name we defined at line 12, and click it. Nothing seems to have happened, but if you look at the console (see box 2), you'll see that all objects are displayed over there.

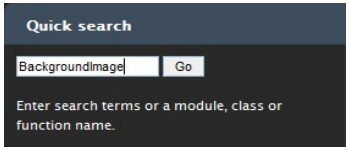
Phase 2 – writing a custom function

You can use [file2.blend] to start from here.

It's time to stop analysing the existing script and write our own functionality. As a first step, remove the content of the main() function. So delete lines 5 and 6.

We want to change the opacity of the background figures, so the first step is to find out where that information is stored. If we look at Blender's User Interface (figure 2), we find BackgroundImage.opacity. So opacity is a property of BackgroundImage. The problem is, where to find BackgroundImage? This can be solved by paying a visit to the API documentation. You can enter your search term at the Quick search box on the right of the page. See figure 3.





At the results page look for the BackgroundImage entry that has bpy_struct between brackets. This means that it is subclassed from

bpy_struct, which is the base class from which all types in Blender are subclassed. This also holds true if you are looking for other types.

When you arrive at the correct page, there are two important places to look for further clues. The first is at the top of the page, where it states the base classes. The second place is at the bottom of the page, where it displays the references. For BackgroundImage there are no interesting base classes, just the basic bpy_struct class. At References you can see that the background figures are stored as part of the SpaceView3D. Follow the link.

At the top of the page you can see that Space is one of the base classes of SpaceView3D. This makes sense as the 3d view space is a type of space.

So let's take a look at the Space entry.

The Space type has no base classes, except from bpy_struct, which we'll ignore. At References we find an interesting bit: Context.space_data. This is interesting, because the context is something we have simple access to. We even already have it included as an argument in the main() function at line 4.

Note: the part between brackets for functions are arguments, for classes it is subclasses.

So let's summarise:

Opacity is a property of BackgroundImage

Background figures are stored in SpaceView3D.

SpaceView3D is a Space.

The current space can be found in the context

Or:

Context -> Space -> Background figures

Now let's turn this into code. Write the following as part of the main() function:

```
for figure in
context.space_data.background_figures:
print(figure.opacity)
```

Run the Script and test it by executing it via the search menu. If you're using the example blend-file it should now have printed 0.5 to the console twice, which is what we're expecting.

The next step is to change the opacity instead of just printing it. Let's say we want to toggle the opacity of the figures between 0.1 and 0.5. So first we check the current opacity and based on that we change it. This is done by using an if/else structure. Replace the print() statement with the following and mind the indentation when you're adding it to your own script.

```
if figure.opacity <= 0.11:
    figure.opacity = 0.5
else:
    figure.opacity = 0.1
```

You might find it strange that we're checking for values smaller than 0.11 (instead of 0.1), but this has to do with floating point accuracy. There is no need to dive into this technical subject, but the important thing to remember

Box3: Python syntax can be confusing when you start. Below you can find a quick list to help you out.

Basic elements

42	Integer
42.05	Float (has decimals)
"42"	String (text, can't be used in calculations)
True, False	Booleans
# 42	Comment, ignored by Python
def():	Define a new function
[1, 2, 3]	List (with 3 integers)
print()	Print the value of a variable

Operations and variables

+	Sum of numbers (floats, integers), or concatenate string
-	Subtract numbers
*	Multiply
/	Divide

All of these can be used in combination with = to do operations on existing variables. So you get:

+=, -=, *= and /=

Example:

```
foo = 7
bar = 6
foo *= bar
print(foo)
# prints 42 to the console (see box 2)
```

Comparing

<	Less than
<=	Less than or equal
>	Greater than

```
>=          Greater than or equal
==          Equal
!=          Inequal

Flow control
foo = 3.14
bar = 42
if foo < bar:
  print("small foo")
else:
  print("big foo")

my_list = [42, “hello”, 3.14]
for i in my_list:
  print(i)

counter = 1
while counter <= 10:
  print(counter)
  counter += 1ole.
```

is that floats aren't 100% accurate. They might have a small deviation. So 0.1 could actually be 0.1000001.

If you test the script it should now already do what you want. We still have some cleaning to do like changing the operator name. You can try this yourself and compare it with the code in [file3.blend].

Phase 3 – Easy access

For following this phase, I highly recommend using [file3.blend] as a base.

We now have an operator that does what we want, but accessing it via the search menu isn't very convenient. Let's add a hotkey for the operator. For simplicity we'll use Q as a hotkey, as it's still free in the default keymap.

Add the following 3 lines to the register() function.

```
wm = bpy.context.window_manager
km = wm.keyconfigs.default.keymaps['3D View']
kmi = km.keymap_items.new("object.change_opacity", 'Q', 'PRESS')
```

You can deduce these lines as we did in Phase 2, by starting with KeyMapItem as a search term. You can also just copy/paste these lines in any script you need. You can use [script2.py] for finding all the internal names of the different keymaps. In our example: '3D View'.

If we run the script now, we can test it by simply using the hotkey **[Q]**.

As a final touch we'll add a button from which the operator can be executed. For this script the hotkey access is much easier, but for some of your own scripts a button might be more useful.

Adding a button is done by appending or prepending a custom function to the draw function of an existing panel. We want to prepend it to the Background Images panel, but for that we need Blender's internal name of that panel. We can find this by opening the User Interface script. Go to **Text >> Open Text Block** and navigate to your 2.57 folder. This is usually located in the folder where you installed Blender. Navigate to **scripts >> startup >> bl_ui** and open the script that contains the panel you need. In our case space_view3d.py.

Doing a search for background, [Ctrl + F], you can find the name of the panel class at line 2199 (might vary for your version of Blender):

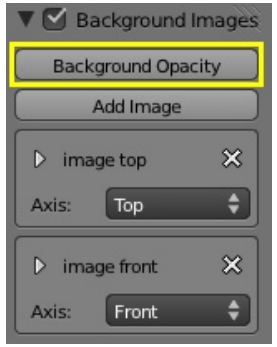
```
VIEW3D_PT_background_figure
```

The next step is to incorporate this into our script. First define a new function which we'll prepend to the panel. Define this above the register() function.

```
def menu_func(self, context):
  layout = self.layout
  layout.operator("object.change_opacity")
```

You can find the different options for the layout at this API page. The final thing to do is prepending this function to the panel. If you look at the API page of the Panel class, you'll notice 3 class methods at the bottom: append, prepend and remove. We'll use prepend() to add our button. Add this to our register() function.

```
bpy.types.VIEW3D_PT_background_figure.prepend(menu_func)
```



We are done now and can use the newly created button to call our custom operator.

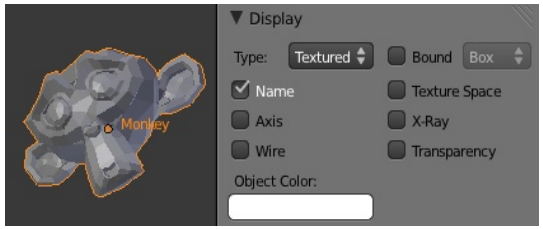
Note: running the script multiple times creates multiple buttons. This can be prevented but goes beyond the purpose of this tutorial, which aims to be an introduction to python scripting. The problem is also pretty minor, as in normal use you'd only run the script once.

Phase 4 – What's next?

You can find the final script in [file4.blend] and as [script1.py].

So what to do now? There are several changes you could make to the script. For instance you could try to make it toggle between 3 values, instead of just 2. You can find an example of this in [script3.py].

If you want to increase your experience, you could also try to write a new script. I'd suggest writing a script to mass toggle the displaying of object names. And for an extra challenge: have it only work on selected objects.



You can take a look at [script4.py] to see a possible way of doing this.

If you get stuck on something while writing your script, there are several resources you can consult. Box 4 displays several of them ●

Links:

There are several useful websites you can use for scripting.

API reference : Essential for scripting, can be accessed from **Help >> Python API** Reference

Python : Official Python documentation

BlenderArtists : Python support forum

For realtime help and feedback you can also join #blenderpython on Freenode at IRC.

Bart Crouch

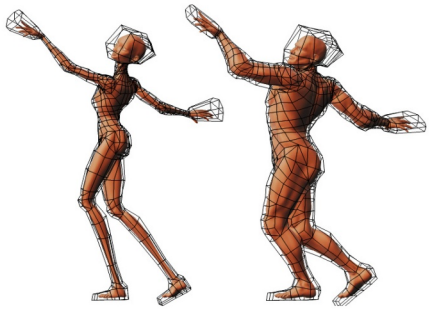
Website:
http://sites.google.com/site/bartiuscrouch/

3D WORKSHOP : BlenRig 4.0 - Auto-Rigging & Skinning system

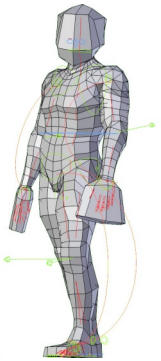
by - **Juan Pablo Bouza**

Introduction

BlenRig is an Auto Rigging & Skinning System for the Blender 2.5+ series created by Juan Pablo Bouza (jpbouza). It features an adjustable rig whose proportions can be changed to fit existing models, or to create a new character based on a previously rigged mesh.

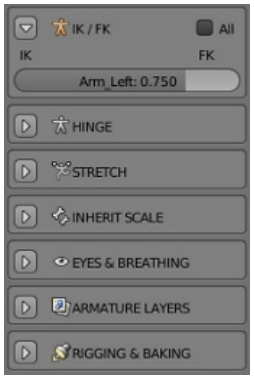


The system consists of an Armature and a strategically modeled Mesh Deform body cage. This allows the user to be able to skin characters with very little or no weight painting at all.



BlenRig MDef Cage

This rig was built up exclusively with Blender's default constraints and drivers, but there is an additional Blender Addon (by Bart Crouch) that can be installed in order to be able to access a dedicated GUI. From here you can directly manipulate the different control modes of the rig (FK, IK, Hinge and Stretchy controls) and several extra functionalities, like the Rig Baking buttons.



Panel by Bart Crouch / JP Bouza.

For the release of BlenRig 4, Juan Pablo created 3 fully rigged character examples. All 3 characters are available for download under a Creative Commons License at: www.jpbouza.com and also at BlendSwap.

Zepam by JP Bouza

Apart from this, the good news is that BlenRig 4 also has 5 video tutorials that can be found here: Or at Youtube and Vimeo.

Human Athletic male by Zbrush artist Nick Zuccarello

Gilgamesh by jpbouza based on a concept by Bassam Kurdali

The Video Tutorials cover:

Addon installation and rig manipulation

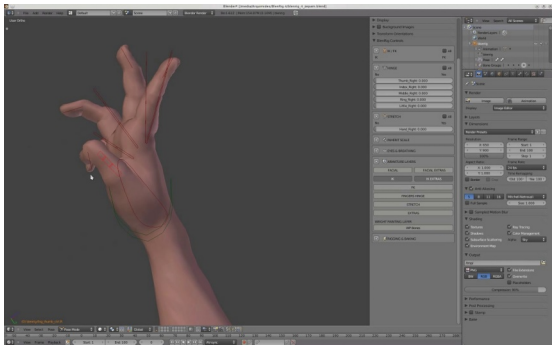
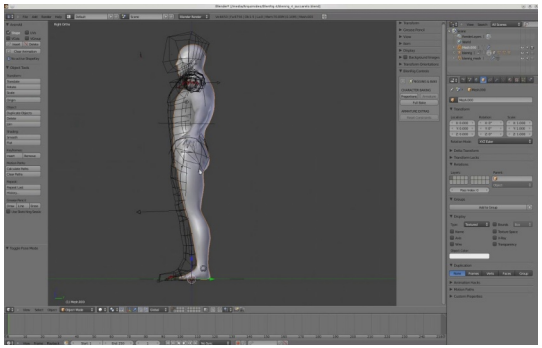


Fig1. Rig controls

3D WORKSHOP : BlenRig 4.0 - Auto-Rigging & Skinning system



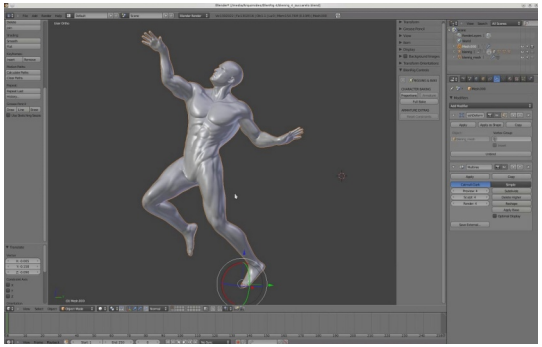


Fig2. & Fig3. Rigging Nick Zuccarello's model.

Rig retargeting and auto-skinning

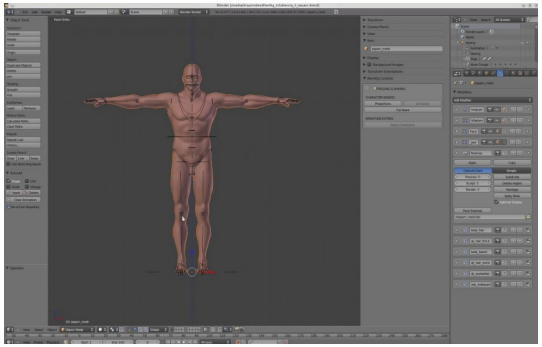


Fig4. Reshaping a rigged model.

Fast Rig and Model re-proportioning, resulting in a new auto-rigged character

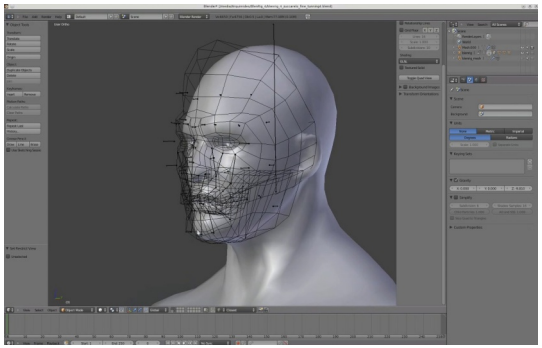


Fig5. Facial rigging.

Totally optional and experimental Facial Rig

The main goals of the BlenRig project are:

* Having a fast and easy way to rig characters

* Delivering correct and complex deformations out of the box

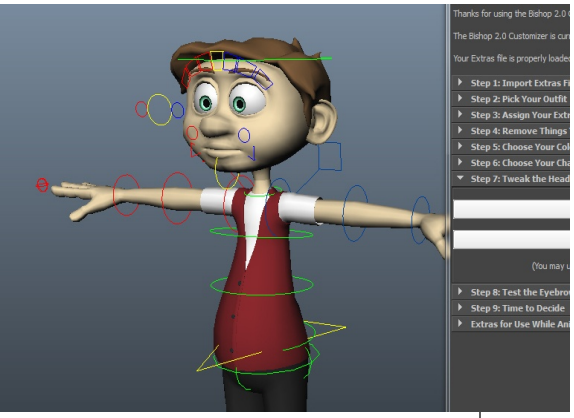
* Having a rig that can provide flexible controls

* Having a user friendly system

Juan Pablo Bouza is Blender User from Argentina. He has periodically released versions of the BlenRig project since 2007.

http://www.jpbouza.com.ars

MAKING OF : How Bishop 2.0 Came to Life



by - **Alexiss Dawn Memmott**

Introduction

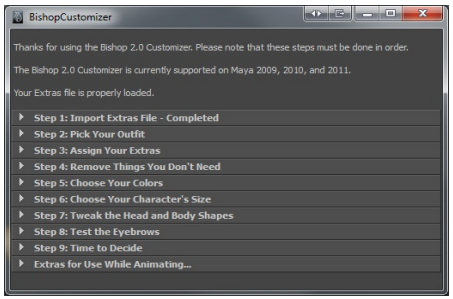
When Animation Mentor first released its Bishop rig to its students in March 2005, he had a fresh new look. Bishop was exciting for industry professionals and students as most animators used a limited stock of free rigs that were available to everyone. Five years — and more than 1,000 graduates later — all of the completed acting tests and short films using Bishop started to give him that dated feel. After all, he was the rig on all student reels and showcases.

Mentor and Student Feedback

We knew an updated rig was in order and as we started the process of envisioning what that would be. But one thing was for sure: The Bishop rig was a super solid rig as Animation Mentor students had spent years animating it. In fact, mentors and students offered constant feedback to make the Bishop rig quite versatile, simple and user friendly. For all these reasons, we decided to build upon what we had instead of reinventing the wheel. That's when Bishop 2.0 came to life.

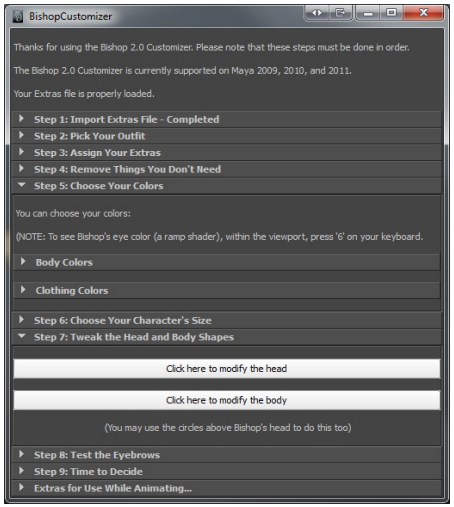
Different Looks

We first talked about morphable rigs and researched different methods on how to best accomplish this — but in the end, we decided to use a combination of blend shapes and deformers.



Step one: Create and setup some basic head and body blend shape targets as a proof of concept. In addition to our team in the office, we worked with Animation Mentor alumnus, Keith Ribbons, to create the first few head and face shapes. We created facial transition tests. Then we created two short tests showing the rig morphing from one rig to another — for example, changing the noses, changing the body shapes and even changing from a woman to a man! Once we created a successful proof of concept, the excitement set in and we got to work on creating Bishop 2.0.

From noses, lips, chins, head shapes, brow shapes, hips, bellies, legs, arms, chests — we created over 100 new targets for each new shape!



Smoke Tests

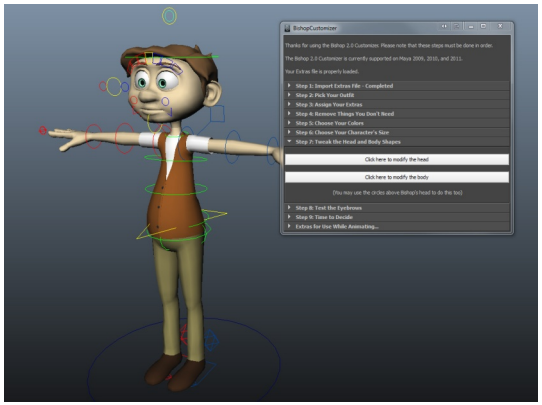
We learned a few valuable lessons along the way — none more important than running "smoke tests." Early on, we would spend a couple of weeks in rig development, only to find that we needed to start from scratch due to a small issue we hadn't caught, such as:

Too many faces. A blend shape target had too many faces and therefore did not match up.

Heavy weight. A new blend shape improperly influenced weighting on one part of the original rig.

MAKING OF : How Bishop 2.0 Came to Life

Super-sized files. The file size balloons and brings the computer to a sudden halt. (But this did produce the aha moment! That is, we learned that importing past geometry included unneeded items such as thousands of light links!)

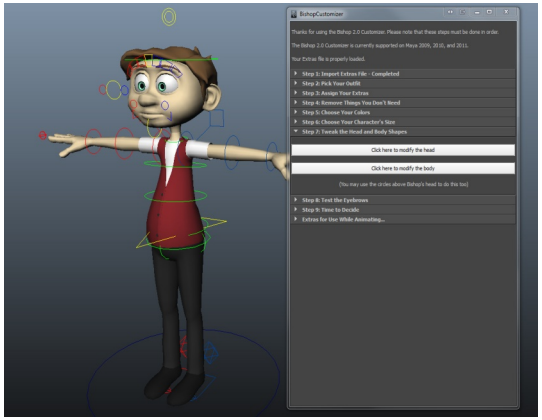


To resolve these types of issues, we created a few different scripts that we ran as smoke tests on the rig. Every day, we would put the rig in a pose that used every control. We wanted to push the extremes to ensure that our animators could confidently pose the rig without the risk of the geometry exploding. Smoke tests are key for anyone in rig development.

Quick, Simple and Worry-Free

We do all of this work so that our students can focus on animation, and not worry about the other elements involved in the pipeline. In rigging, it's easy to add a ton of controls to account for every possible option — but this makes the rig clunky, slow and confusing to use. Our number one goal was to create a rig that would open up various new character designs for our students, maintain the ease of use of the rig and make the new process of creating a character quick and simple.

Accomplishing the goal also brought on new challenges, such as keeping the file size down. Each article of clothing and blend shape target contributes to the file size. Through a series of scripts, and with the help of our partners and friends at AnimationRigs.com, we created a solution that guides the user through the character creation process. Here, students can resource a single panel to create a new character and be ready to animate in just a few minutes.

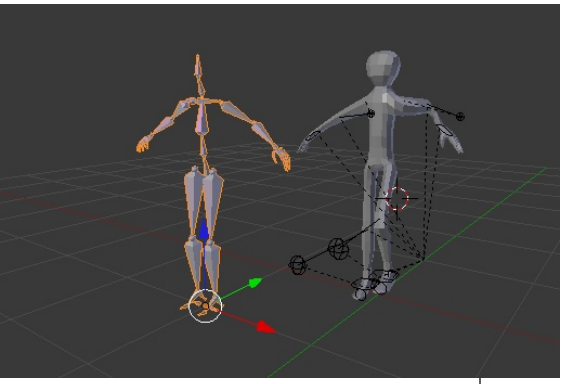


With Bishop 2.0, we did not create a new rig from scratch. We added to the solid base that we had already built. The new blend shape targets, clothing geometry, wigs and glasses have opened up the possibilities for various character designs for our students and alumni. And all while enhancing an already great user experience with respect to file size, loading times, ease of use and simplicity combined with versatility ●

Alexiss Dawn Memmott – bio

Alexiss is the content administrator and an animator at Animation Mentor in Emeryville, California. She started her university studies at Brigham Young University where she received a bachelor's in Psychology in 2006 and then started as a student at Animation Mentor the following January. After finishing her student short film 18 months later, she joined the team at Blue Sky Studios as an animator to work on Ice Age: Dawn of the Dinosaurs. In January 2009, she moved to Lumenas Animation Studios to work on The Legend of Santa Claus.

KNOW HOW : Modding Rigify for Stretchy Arms & Legs



by - **Oscar Baechler**

Introduction

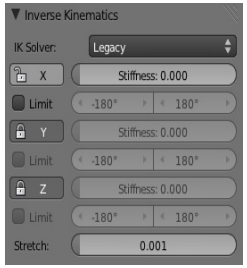
Nathan Vegdahl's Rigify turned the Blender world upside down. With the addition of an auto rigging macro, a process that took expert riggers days or weeks can now be reduced to a matter of hours by a n00b. So what's the next frontier of rigging? Pimping your Rigify rig! Already a number of tutorials have surfaced, showing how to mod Rigify to include face rigs, animal parts, etc.

I used Rigify as a starting point for my Cataphract rig, before modifying it to have lots of crazier anatomy: multiple heads, a tail, squash-and-stretch splines for the necks, spine and tail, and inverse kinematics (IK) functionality for the neck, spine and tail.

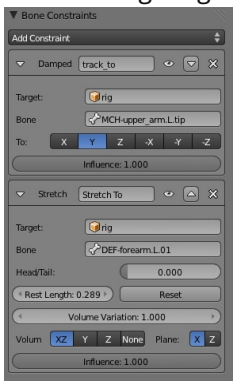
Most projects won't need that level of cartoony rigging. However, a quick change to the default Rigify rig can at least get you rocking out to some stretchiness on your arms and legs. The process is tedious, but thankfully short!

Start by using Rigify in the default manner. Add a human meta rig, and in edit mode change the bone locations around until they fit your character's frame. Once that's done, on the Object Data properties panel, click "Generate." You've got a rig all set and ready to use. Now let's make its arms and legs stretchy!

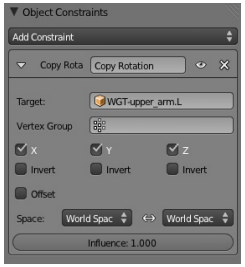
1. On rig layer 19, select MCH-shin_ik.L. Under the bone properties menu in the Inverse Kinematics tab, change the stretch to .001. Repeat for every IK bone: shin, thigh, arm, forearm.



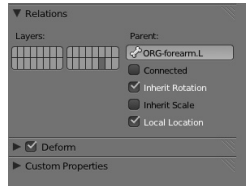
2. On rig layer 18, select DEF-upper_arm.L.01. Delete the Copy Location and Copy Scale constraints. Add a StretchTo constraint. Target:rig. Bone:DEF-forearm.L.01 Head/Tail:0, Volume Variation 1, volume XZ,and Plane of X. Repeat this process with DEF-upper_arm.R.01, DEF-thigh.L.01 and DEF-thigh.R.01 (but using their relevant DEF-forearm or DEF-shin Stretch To target)



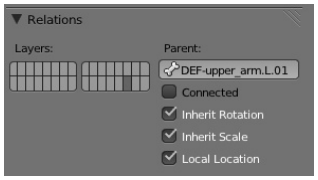
3. Select DEF-upper_arm.L.02. Go into Edit Mode, and under the Bone Properties tab, change its parent to DEF-upper_arm.L.01. Add a Copy Rotation constraint, with target: rig and bone: ORG-upper_arm.L. Constraint should affect X, Y and Z with Worldspace/Worldspace selected. Repeat this with DEF-upper_arm.L.02, DEF-thigh.L.02, and DEF-thigh.R.02, using their equivalent new parents and constraint targets. (in other words, the Copy Rotation constraint on DEF-thigh.R.02 is ORG-thigh.R)



4. Select DEF-forearm.L.01. On the Bone Properties panel under the Relations tab, turn Inherit Scale OFF. Add a Copy Rotation constraint with target: rig, bone: ORG-forearm.L, affecting X/Y/Z in Worldspace/Worldspace. Add a Stretch To constraint, target: rig, bone: Def-hand.L. Set the Head/Tail to 0, Volume Variation to 1, Volume to XZ and Plane to X. Repeat



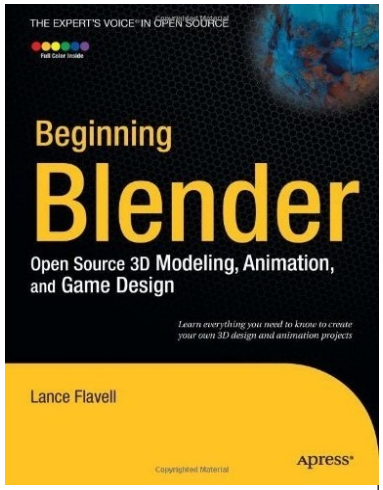
this on DEF-forearm.R.01, DEF-shin.L.01 and DEF-shin.R.01, but with relevant targets. In other words, DEF-shin.L.01 will copy the rotation of ORG-shin.L and Stretch To DEF-foot.L.



5. Select DEF-forearm.L.02. In edit mode, change its parent to DEF-forearm.L.01. On its Damped Track constraint, change the target bone to ORG-hand.L. Repeat this process with DEF-forearm.R.02, DEF-shin.L.02, and DEF-shin.R.02 but with relevant changes. In other words, DEF-shin.L.02's new parent will be DEF-shin.L.01, and its Damped Track constraint's target bone will be ORG-foot.L.

That's it! If you are unfamiliar with Rigify you may have trouble identifying where the changes happened. Turn on the IK bones and switch their FK/IK attributes to 1. Now, when you move your IK controllers around, the deform bones stretch to meet them!

BLENDED REVIEW : Beginning Blender



by - **Sandra Gilbert**

From the introduction:

... a guide to explain just the few important options needed to get me started.

"This book does not aim to be exhaustive and yet it is not written to an overly simplified manner so as to insult your intelligence. 3D animation by its very nature is not simple. What you have with Beginning Blender is a book that covers a good range of the many different areas of Blender, with practical examples to get you fast-tracked into using those areas."

Lance has produced a great introduction to Blender, filled with tips, tricks and techniques guaranteed to get you up to speed with Blender's most important tools, options and settings. Lance uses a great style of writing that gives the reader a good foundation for further exploration and study in Blender.

Each chapter focuses on a different area of Blender and guides you to the most important tools and options. There are tables, diagrams and call out areas showing key information that can be quickly referenced. The examples are easy to follow and show how easy it is to accomplish a variety of common and a few not so common tasks in Blender.

In this review I am going to focus on the two rigging chapters. It is amazing how much information Lance packed into these two chapters alone. And even if for some bizarre reason you weren't interested in the rest of the book, these two chapters would still make it worth owning.

Chapter 7 covers basic rigging and animation. Lance covers keyframing, the dopesheet (which was cool because I hadn't played with that yet), parenting, the graph editor, pivot points (restricting movements), basic tracking, bone explanations and basic rigging of a character.

One of the examples for this chapter teaches basic tracking by showing you how to set up “Eyes that Follow”. Now this is a right handy example and something that you will find yourself referencing for most if not all characters.

When you reach the “Rigging a Simple Character” section, Lance walks you through a complete (simple) rig set-up including weight painting and bone envelopes. There are clear full color screen shots showing what you should have at each stage which makes this chapter a very helpful reference for future rigging experiments of your own. Along the way he shares some useful rigging tips that make animation easier, such as putting a little bend in joints like elbows and knees.

Once you have gotten comfortable with basic rigs, it is time to move onto chapter 8, “Advanced Rigging”, where Lance ramps it up a notch.

First off he explains Forward and Inverse Kinematics and their uses. There are great examples in this chapter, starting off with an IK arm. You are shown how to set up the arm and then walked through setting up constraints to make it behave properly.

Once your arm is set up, Lance then shows you how create and set a custom bone shape for your new IK arm. We look advanced already.

Next up, there is a walk through of an IK Leg and then a Reverse Foot rig. Now I always liked the Reverse Foot rig, but can never get it set up right without step by step instructions. I probably still can't even after reading this (my issue, not Lance's teaching), but no worries, now I have clear steps and screen shots to follow. Single Bone Finger Controls are next up on the hit parade with the same attention to detail as the previous examples. He makes the Single Bone Finger Controls look super easy to set up and even

BLENDED REVIEW : Beginning Blender

easier to animate once made.

These examples make easy work of creating some of the more tricky and advanced rig components and are covered in a way that makes it easy to flip back and reference them as needed. Which in my case, is every single time I need to build a rig.

Rigs of course are not overly useful if you don't know how to animate, so Lance takes you through a complete walk cycle. I liked how he sets up his walk cycles and the tips he gave along the way, especially the little details, like peeling of the feet.

The whole last half of the chapter is devoted to shape keys and how to use them to create facial expressions and lip syncing. He has some good images of the shapes created showing the most common shapes needed for a wide range of emotions and shapes for lip syncing.

Beginning Blender will be sitting on my desk for some time to come. The way it is set up makes it a good reference guide to a number of common tasks that manage to slip my mind while engrossed in whatever my latest project is

Beginning Blender: Open Source 3D Modelling, Animation, and Game Design

by: Bethany Hlitola

Paperback: 448 pages

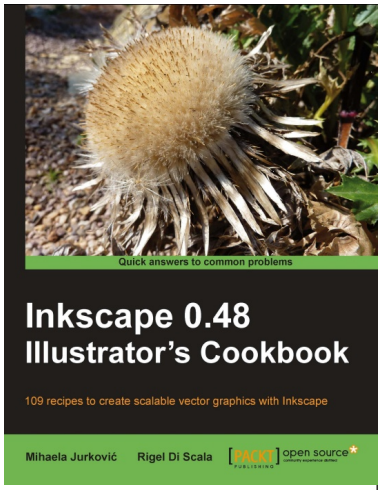
Publisher: Apress; 1 edition (December 30, 2010)

Language: English

ISBN-10: 1430231262

ISBN-13: 978-1430231264

BLENDED REVIEW : Inkscape 0.48 - Illustrators Cookbook



by - **Gaurav Nawani**

Inkscape is an excellent vector tool for a graphic designer. While it may have been true that like all OSS softwares the popularity and adaptation was restricted due to non availability of information available. Situation for Inkscape was no different but now we have lots of books available from different publishers.

The recent book to come out of Packt publishing's stable is 'Inkscape 0.48 Illustrators Cookbook written by Mihaela Jurković and Rigel Di Scala.

This is an unusual choice of book title 'Illustrators Cookbook', it would appear that the book is targeted specially at illustrators by the means of topic covering graphic design for illustrators but not quite so this book is generic in approach towards Inkscape's tool set.

The manner in which the book is prepared is like small how to's for every feature or tool then progressing with a interesting and tougher topic, although the text is mostly easy to follow, does makes it a bit monotonous to read in one go. Surprisingly the book was missing a screen-grab of the application in question.

Good thing is whatever this book does it does it thoroughly. The choice of topics for chapter is again keeping in line with increasing complexity of tools/features.

Chapter 1 Editing Objects: covers objects shapes, pen tools and calligraphic controls etc

Chapter 2 Editing Colors: covers colors, strokes and editing gradients.

Chapter 3 Speeding up your work flow: covers all those support tools in a vector design tool illustrator can't leave without, like align distribute, grid, guide and snap etc.

Chapter 4 Creating and editing clones: This was the chapter what most intermediate users will be reading diligently, it covers clones. tiled clone and creating patterns out of tiled clones.

Chapter 5 Live path effects: Another delightful read for getting into the detail of powerful feature set available in Inkscape to manipulate paths, one of the most significant toll in any vector drawing tool.

Chapter 6 Extension: It covers various tools such as Lsystem generator, 3drendering and barcode generator etc.

Chapter 7 till 11 the book continue with additional features and recipes. There is plenty of action to learn from those.

There are various small how to dos which excite like patterns, using the new 3d box tools and rail sleepers to name a few. They give you something to look forward to in each chapter.

So who is this book for you might ask, through book mentions its for intermediate to expert I would put it at beginner to intermediate reason being that this book presents itself in a manner more suitable for learn and refrence.

Its like a compendium of Inkscape 0.48 or help book manual if you will which can show you how to use this and that feature effectively. Just one thing of caution before you jump right in, do make sure you can navigate Inkscape on your own to be able to use theis knowledge effectively.

More power to Inkscape ●

★★★☆☆

Inkscape 0.48 Illustrators Cookbook

Language : English

Paperback : 340 pages [235mm x 191mm]

Release Date : April 2011

ISBN : 1849512663

ISBN 13 : 978-1-84951-266-4

Author(s) : Mihaela Jurkovic, Rigel Di Scala

Want to write for BlenderArt Magazine ?

Here is how

Step1. Choose what you want to write.

- Tutorials explaining Blender features, 3dconcepts, techniques or articles based on the focused theme of the issue.

- Reports on useful Blender events throughout the world.

- Cartoons related to blender world.

Step2. Send submissions to sandra@blenderart.org.

- Send us a notification on what you want to write and we can follow up from there.

Step3. Some guidelines you must follow.

- Images should be properly cut and represent the text appropriately.

- Images should be provided seperately in a folder named (images, img or pictures).

- Images should be named/labled likewise (image1 or img1 etc).

- Provide proper captions for images if and when needed.

- Image format prefered is PNG but good quality JPG can also do.

- You can submit inline images in documents like DOC or Openoffice ODT etc but make sure the images were properly names before importing them in docs.

- Images inside a PDF are a strict no, but a pdf document with images if provided to show how the author wants the formating of doc will be appreciated.

- Make sure that screenshots are clear and readable and the renders should be at least 800px, but not

- Text should be in either ODT, DOC, TXT or HTML.

Step4. Archive them using 7zip or RAR or less preferably zip.

Step5. Additional stuff that you can do

- Please include the following in your email:

- Name: This can be your full name or blenderartist avtar.

- Photograph: As PNG and maximum width of 256Px. (Only if submitting the article for the first time)

- About yourself: Max 25 words .

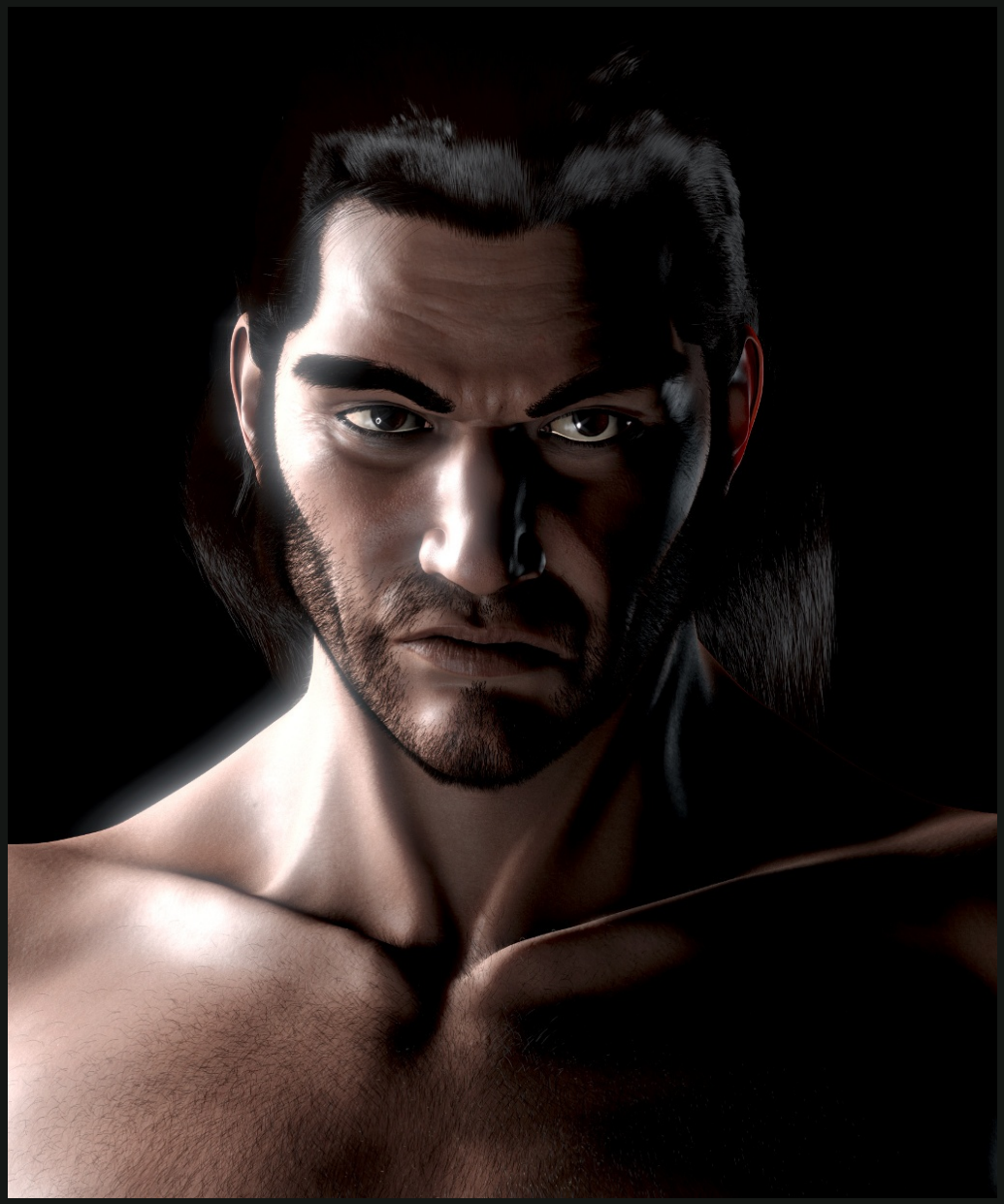
- Website: (optional)

Note: All the approved submissions can be placed in the final issue or subsequent issue if deemed fit. All submissions will be cropped/modified if necessary. For more details see the blenderart website.

BA takes no responsibility fo the material in any form and the submission will automatically mean that you have agreed to the blenderart terms and conditions for submission for more information please do read the disclaimer.

Chimneys - by Darcy Brooker

Zepam -by Juan Pablo Bouza



Old man -by Manollo